May 1988 LIDS-P-1775

AN ADAPTIVE DISTRIBUTED DIJKSTRA SHORTEST PATH ALGORITHM

Pierre A. Humblet

ABSTRACT

We give a distributed algorithm to compute shortest paths in a network with changing topology. It does not suffer from the routing table looping behavior associated with the Ford-Bellman tdistributed shortest path algorithm although it uses truly distributed processing. Its time and message complexities are evaluated.

Pierre Humblet is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge MA 02139.

This research was supported in part by Codex Corporation and in part by the Army Research Office under Grant No. DAAL03-86-K-0171.

1) INTRODUCTION

One of the oldest and best known problem in the field of distributed algorithms is to compute shortest paths between nodes in a network. This problem arises in the following context. We have a network of links and nodes (processors). Each link (I,J) is characterized by a direction dependent length LEN(I,J) that can change with time and can only be observed at node I. The nodes execute a distributed algorithm to keep track of the shortest distances between themselves and the other nodes, in the course of which they communicate with adjacent nodes by transmitting messages over the links. The most popular solution to this problem is the Ford-Bellman method that was originally introduced in the Arpanet [McQu-74] and is now used in a large number of networks [Burr-81], [Cegr-75], [Dec-83], [Jubi-85], [Spro-81], [Taji-77], [West-82], [Bert-87], [Schw-86]. It basically works as follows:

Two kinds of information are maintained:
- the Routing Table RT_D(I,J), whose (I,J) th entries are maintained at node I to contain the estimate of the minimum distance between I and J.
- the Neighbor Table, NT_D(I,J,P), where the first two indices are node identities and the third is a link adjacent to the first node. If P= (I,M) NT_D(I,J,P) is used to save at I the latest value of RT_D(M,J) transmitted by M to I.

The algorithm consists of the following steps:
-Initially RT_D(I,J) is set to 00 for all J, except RT_D(I,I) which is set to 0, and all links are Down.
-Whenever a link adjacent to I goes Up, node I sends records of the form (J, RT_D(I,J)) over it, for all nodes J.
-When a node I receives a pair (J,D) over a link P, with I # J, it sets NT_D(I,J,P) to D and it computes RT_D(I,J) = min (over p) NT_D(I,J,p) + LEN(p). If this results in a new value for RT_D(I,J), the record (J,RT_D(I,J)) is sent to all the neighbors of I.
-The same computation is also performed at I for all nodes J not equal to I whenever the length of any adjacent link changes. In particular, the length of a Down link is considered to be infinite.

This basic algorithm and a number of variations have been shown to converge to the correct distances if the link lengths stabilize [Taji-77], [Hago-83], [Bert-87] and all cycles have strictly positive length. However the convergence can be very slow when link lengths increase. In a typical example (figure 1) node 1 becomes disconnected. Nodes 2 and 3 keep executing the algorithm, increasing their RT_D(.,1) without bound. This behavior is known as "counting to infinity". While this goes on messages destined to 1 may cycle back and forth between nodes 2 and 3, a phenomenon called "routing table looping". In practice there are known upperbounds NN on the number of nodes and MAXLEN on LEN() and entries

of RT_D that exceed (NN-1) * MAXLEN are set to 00. If not all Up links have the same length, a better alternative [Dec-83] is to keep track of the number of links in a shortest path and to only accept paths up to a maximum number of links.

The looping behavior problem is a major drawback of Ford-Bellman distributed algorithms (for an analysis in a simple case, see [John-84]). To prevent it [Jaff-82] extends techniques developped in [Merl-79] and "freezes" part of the network while the news of an increase in length propagate. This approach requires new types of messages and sometimes delays a node from obtaining a correct distance. It is also discussed in [Gruch-?] and [Schw-86]. Another approach [Garc-87] reduces the likelihood of looping but, in our opinion, does not always prevent it.

In this paper we offer a novel algorithm that can be seen as a distributed Dijkstra algorithm and we analyse its behavior. Its major advantage is that it does not rely on "counting to infinity" and it does not suffer from routing table looping.

2) DESCRIPTION OF THE DISTRIBUTED DIJKSTRA ALGORITHM

It has often been noted that in the previous algorithm the RT_D(I,J)'s for different J's behave independently of each other and that one can focus on a single destination. To the contrary we remark here that much can be gained by considering the interactions between different destinations.

Assume we know the neighbor K next to the destination on the shortest path from a node I to a destination J. The following statements must hold if we have valid paths to K and J and 0 <= LEN() <= MAXLEN:

- a neighbor of I that lies on a shortest path from I to J must also lie on a shortest path from I to K.
- RT_D(I,J) >= RT_D(I,K)
- RT_D(I,J) <= RT_D(I,K) + MAXLEN

This suggests that keeping track of the nodes next to the destinations (on shortest paths) is useful (this is different from keeping track of the next node on a path, which is only marginally effective [Cegr-75], [Schw-80], [Spro-81]). Although the previous relations could be used to quickly weed out unreachable nodes in Bellman-Ford type algorithms and prevent routing table looping, we will not use them directly in the rest of the paper. Rather, we note that keeping information at a node I about the nodes next to destinations is equivalent to keeping track of an entire shortest path tree rooted at I. This is the view that we will adopt and exploit to develop a new shortest path algorithm. In the next subsections we introduce the data structures and describe the algorithm . This will be followed by sections on the proof of

correctness (3), efficient implementations (4), complexity measures (5) and finally comparisons with other methods.

2.1 Data Structures:

Our goal here is to keep track of an estimated shortest path tree at each node, and the "replica" of such a tree at the adjacent nodes. Although this could be done quite abstractly, we prefer extending the simple and explicit notation already used in section 1.

- To keep track of a shortest path tree at a node I we use three kinds of Routing Table entries, RT_D, RT_N and RT_L. RT_D(I,J) is as before; RT_N(I,J)denotes the node next to J on the shortest path from I, while RT_L(I,J) indicates the link adjacent to I on a shortest path to J.
Note that entries RT_N(I,I) are meaningless and they will never used. In contrast NT_N(M,I,P) is always M if P = (M,I).
- The Neighbor Table now contains two kinds of entries, NT_D and NT_N. NT_D(I,J,P) is as before while NT_N(I,J,P) is meant to be the node next to node J on a shortest path to J from node I, via link P.
- Messages sent by a node I consist of packets of records, each record being a triple of the form (J,RT_D(I,J),RT_N(I,J)).

2.2 Algorithm:

The detail sof the implementation appear in figure 2. The algorithm is composed of two major parts: in the first part, a node observes local topology changes or receives update messages from neighbors; these updates are saved in NT. In the second major part (COMPUTE) each node I builds from NT a large tree with weighted edges (figures 3 and 4), where a node identity may appear many times: node I puts itself as the root and "hangs" on each adjacent link the shortest path trees communicated by its neighbors. This large tree is then scanned in a "breadth first" fashion (with respect to the cumulative edge weights from the root) to obtain a subtree where each node appears at most once. That subtree is adopted as the new "local" shortest path tree and changes (if any) with respect to the previous version are communicated to the adjacent nodes.

More precisely, COMPUTE() at node I builds RT_D and RT_N starting with I, considering nodes in order of nondecreasing distances from I, and including a node in RT only if it has not been included yet and if its neighbor toward I in NT already is in RT. Thus the RT structure forms a directed tree (this would hold even if the NT's did not form trees) that is composed of a root node out of which subtrees from the NT's hang. We will call that tree the Routing Tree.

The description of figure 2 leaves vague exactly when COMPUTE() is executed, specifying only that it is executed within a finite time after a triggering event. Concrete possibilities will be suggested in section 5.

Because it uses a breadth first search (with respect to total length), our algorithm can be seen as an adaptive distributed version of Dijkstra's algorithm [Dijk-59]. Another distributed but static implementation of Dijkstra's method has been given by [Frie-79]. These approaches should not be confused with those relying on an explicit topology broadcast followed by local computation [Rose-80].

3) PROOF OF CORRECTNESS

For the algorithm to work some assumptions on the behavior for the links and nodes must hold. They are similar to those used by other authors:

1- There is a link protocol that maintains Up and Down states for the links and handles message transmissions. It has the following properties:

a) A time interval during which a link is Up at a node is called a Link Up Period (LUP) at that node. To a Link Up period at one end of a link may only correspond at most one LUP at the other end. Both ends on the link can only remain in non-corresponding LUP's for finite time intervals.

b) A message can only be sent during a LUP at the source, and it can only arrive during a corresponding LUP.

c) Messages transmitted over a link during a LUP either arrive intact, within a finite time, in the order they were sent, or they never arrive. If a message never arrives the LUP at the source node must be finite.

2- Nodes can similarly be Up or Down. There are no LUP's at a node while it is Down.

3- All nodes are initially Down.

4- All update and link state change messages received on (or about) a link are processed one at a time in the order they are received.

5- Link lengths are strictly positive numbers (*). A Down link is assigned length 00.

6- Between times 0 and T links and nodes go up and down and link lengths change, but at time T links have the same status at both ends and there is no change after T. This assumption is only made to allow us to prove that the algorithm converges after time T to correct values.

****

(*) 0 link lengths can be allowed (even 0 length cycles) by the following artifice: replace them with a positive length equal to the smallest strictly positive length divided by the number of nodes. This will not affect shortest paths. Alternatively the algorithm can be modified to directly handle 0 length links by considering that if paths A and B have the same length but A contains fewer links than B, then A is "shorter" than B.

Below we will use the word final to refer to the link lengths, shortest paths, etc.. in the topology after time T. D(I,J) will denote the distance between nodes I and J in that topology.

It is easy to see (we omit the formal proof) that assumptions 1-4 together with the algorithm of figure 2 guarantee that values of RT_D(I,.) and RT_N(I,.) at and after time T will eventually be reflected in NT_D(M,.,P) and NT_N(M,.,P) if P=(M,I) (except for the peculiar behavior of RT_N(I,I) and NT_N(M,I,P) alluded to in 2.1). We will show that the algorithm is correct, in the sense that within a finite time after T the RT structures at all nodes will form final shortest path trees. The proof consists of two parts: the first part shows that information about the old (before time T) topology is eventually flushed from the network. The second part shows that shortest path trees are eventually computed.

Let T(0) = T and for a given execution of the algorithm let T(k+1) be the time by which all messages that are in transit at time T(k) (k > = 0) have arrived at their destinations and have been processed (including running COMPUTE()).

Theorem 1: By time T(K), all paths from the root in a Routing Tree that have no more than K links have their final length.

Proof: The theorem is true at time T(0) for K = 0. Assume that the theorem is true for all K < N (N > 0). By time T(N), the routing trees at time T(N-1) of all nodes have been communicated to their neighbors. A path with no more than N in a routing tree is the concatenation of an adjacent link (which has final length from time T(1) on) and a path with less than N links in the Routing Tree of an adjacent node, and thus it has a final length from time T(N) on.

The theorem does not say that at time T(K) paths with no more than K links are shortest paths. This last statement only holds at later times; specifying when requires new notation.

Definitions:
Let H(I,d) be the maximum number of links in loop free directed paths starting at node I and having length not exceeding d in the final topology.

For given nodes I and J define S(I,J) and L(I,J) as follows:
S(I,J) = the set of neighbors of I on a final shortest path to J
L(J,J) = 0
$$L(I,J) = \max\left(H(I,D(I,J)),\ 1 + \max_{K \text{ in } S(I,J)} L(K,J)\right)$$

That the definition of L() makes sense follows from the fact that the set of links on shortest paths to node J forms a directed acyclic graph so that the L(I,J)'s can be defined iteratively.

Theorem 2:

1) If J is not connected to I in the final topology, RT_D(I,J) = 00 for all times after T(H(I,00) + 1).
2) If J is connected to node I, the Routing Tree at node I includes a final shortest path to node J from time T(L(I,J)) on.
3) If paths included in the Routing Tree are selected with tie breaking rules that depend only on shorter paths stored in the NT's (i.e. the rules are not time varying, or random, or depending on irrelevant longer paths) then the shortest path to node J in the Routing Tree at node I will not change from time T(L(I,J)) on, and thus the algorithm will terminate in finite time.

Proof: The first part of the proof follows directly from theorem 1 and the definition of H(I,d).
The second part is proven by induction on the distance from I to J. It is true at time T(0) at node J. Assuming it is true at time T(L(K,J)) for all nodes K that have D(K,J) < D(I,J), we will show it will hold at node I at time T(L(I,J)).
Theorem I insures that by time T(H(I,D(I,J))), all paths of length not exceeding D(I,J) include only links with final lengths, thus the Routing Tree cannot contain a path to J of length less than D(I,J).
By time max (over K in S(I,J)) T(L(K,J) +1) a final shortest path from all neighbors of I on a final shortest path from I to J will be reflected in the NT at node I and COMPUTE() insures that the Routing Tree of node I will include a path to J. (The "max" is needed as the statement of COMPUTE() does not indicate how ties are broken in selecting P*).
Similar induction shows that under the hypothesis in the third part the path to J in the the Routing Tree at I will not change after time T(L(I,J)).

4) EFFICIENT IMPLEMENTATIONS

The facts that COMPUTE() involves sorting nodes and that messages include identities of nodes next to destinations may seem prohibitive. We indicate here how simple data structures can alleviate much of the difficulty. Below, NN denotes the number of nodes in the network and A(I) the number of links adjacent to node I.

To avoid the sorting operation, nodes in a Neighbor Table can be organized as a doubly linked list, in order of increasing NT_D. Notice that COMPUTE includes records (J,D,K) in an Update message in order of non-decreasing D so that the linked list for NT can be updated with an amount of processing not worst than linear in NN. Running COMPUTE() at a node requires than an amount of processing not

worst than linear in NN * A(I) * log(A(I)) as there are a total of NN * A(I) entries in the NT linked lists and determining P* during a step can be done in time proportional to log(A(I)).
Also in a record (J,D,K), node K must appear before J in the updated NT list. Thus the identity of K can be encoded as a number e.g. specifying the position of K in the list. This can make significant savings in networks where node identities are character strings.

A more efficient (and complex) implementation is to keep a direct representation of trees for RT and NT. When a new RT is computed, only the difference between the new and old trees need to be communicated, e.g. as a set of subtrees. Recall that a subtree of N nodes can be transmitted as N node identities plus 2N bits. This can be done by walking along the subtree in depth first fashion, transmitting a node identity the first time it is met, transmitting a 0 bit each time a link is traversed away from the root, and a 1 bit when a link is traversed toward the root. If this is done, updating NT takes an amount of time proportional to the number of nodes in an update message.

Other savings can be realized by using information specific to each network. For example in networks where link lengths change by relatively small amounts it is likely that the structure of the Routing Tree will often remain unchanged despite some link length changes. It is easy to design coding schemes taking advantage of this feature.

Various optimizations are also possible. For example when receiving an update message one can easily determine if COMPUTE() needs to be run. Also a node I need not send updates about a node J to an adjacent node K while J is in the subtree below K in the Routing Tree at node I.

5) TIME AND MESSAGE COMPLEXITIES

We define the time complexity of the algorithm as the largest time that can elapse between the moment the last topology change occurs and the moment all nodes have final shortest paths to all other nodes. The message complexity is defined as the maximum number of node identities exchanged during that same period.

Before evaluating the complexity of the algorithm, we must precise when COMPUTE() is executed after a triggering event in part 1 of figure 2. There are two traditional possibilities, and we also suggest another:
A) event driven: run COMPUTE() whenever a topology change occurs or an update message is received. One expects that this would be the fastest. However if the output links have finite capacity this might result in update messages queueing for transmission.
B) periodic: run COMPUTE() at each node on a periodic basis, the periods need not be the same at all

nodes. This has the effect of delaying propagation of changes, but may reduce the computational load and the number of transmitted messages.
C) The third possibility combines the advantages of A) and B): use A) but avoid the possible queueing of messages by combining all messages queued on a link into a single one. That message indicates all the changes that must be made to NT at the receiving end in order to obtain there the image of the current RT at the source.

If the algorithm is operated in an event driven manner, little can be said about the time necessary for the algorithm to complete, or about the number of messages that need to be exchanged. Examples like the one in [Spin-87] can be devised to show that the number of messages may grow exponentially with the number of topology changes.

More can be said if we operate following B) or C) and use as time unit an upperbound on the time between the moment a message is generated and the moment it is processed (including running COMPUTE()). The results stated in theorem 3 then follow from theorem 2. Below MaxHop(I,J) and MinHop(I,J) denote respectively the maximum and minimum number of links in shortest paths between nodes I and J, while R denotes the ratio of the largest to the smallest values of link length assigned to an Up link. R is often less than 2 or 3 in practical situations.

Theorem 3: If T= 0 and messages are processed within one time unit after they are generated, then
- If I is not connected to J, RT_D(I,J) = 00 by time H(I,00) + 1
- If J is connected to I, RT at I includes a shortest path to J by time:
1.1) L(I,J)
1.2) min (NN + MaxHop(I,J), R * MinHop(I,J))
1.3) min (NN + MinHop(I,J), R * MinHop(I,J)) if COMPUTE() is modified to break ties in favor of the path with fewer links in selecting P*
2.1) MaxHop(I,J) if just before time T all path lengths stored in NT's and contained in messages in transit are not less than the final lengths (e.g. if all nodes where isolated just before time T)
2.2) MinHop(I,J) under the assumptions in 2.1 and 1.3

Proof: The first statement and that in 1.1 follow directly from theorem 2. Note that H(I,00) + 1 never exceeds NN.
1.2 follows from 1.1 and from using the facts "H(I,d) < NN for all d" and "H(I,D(I,J)) <= R * MinHop(I,J) " in the definition of L(I,J).
1.3 follows by examining why the Max operation is used in Theorem 2.
2.1 follows by noticing that under the hypothesis in 2.1, H(I,D(I,J)) plays no role in Theorem 2.

2.2 follows from 2.1 as 1.3 follows from 1.2.

Regarding the communication complexity, we can make a statement if one assumes both that all messages are processed within one time unit after being generated and that at most one message can traverse a link within a time unit. Those can be realistic assumptions in cases B) and C). The time bounds of theorem 3) can then be transformed in bounds for the communication complexity: it does not exceed a function linear in NN * NL * min(NN,R * Diam), where NL denotes the number of links, and Diam is defined as the maximum (over I and J) of MinHop(I,J).

6) COMPARISON WITH OTHER METHODS

Both our algorithm and distributed Ford-Bellman perform equally well and are at their best under the assumption in part 2.1 of Theorem 3, i.e. when all estimated distances are initially too large. Under general conditions ours performs better, as it does not "count to infinity". More importantly from an operational viewpoint, it does not suffer from "routing table looping". In addition the complete sequence of nodes on a shortest path can easily be derived from the RT structure. Including this sequence in packets is an easy way to guarantee that that absolutely no looping will occur, which is very desirable for systems using virtual circuits.

Shortest paths can also be computed by broadcasting the topology and performing local computation [Rose-83]. This approach typically is faster and requires fewer messages. However it requires more storage and processing is not distributed: each node computes its Routing Tree independently, while in our approach a node benefits from the computation done by the neighbors. The difference is striking in the case nodes that have only one adjacent link. Although we prefer the topology broadcast method if enough memory is available, we advocate the algorithm presented in this paper as a migration path for networks that currently use Ford-Bellman. Our method uses similar data structures and messages, but it does not suffer from routing table looping.

Acknowledgements

Thanks to Steve DiCecco for reawakening our interest in the shortest path problem and for providing constructive comments.

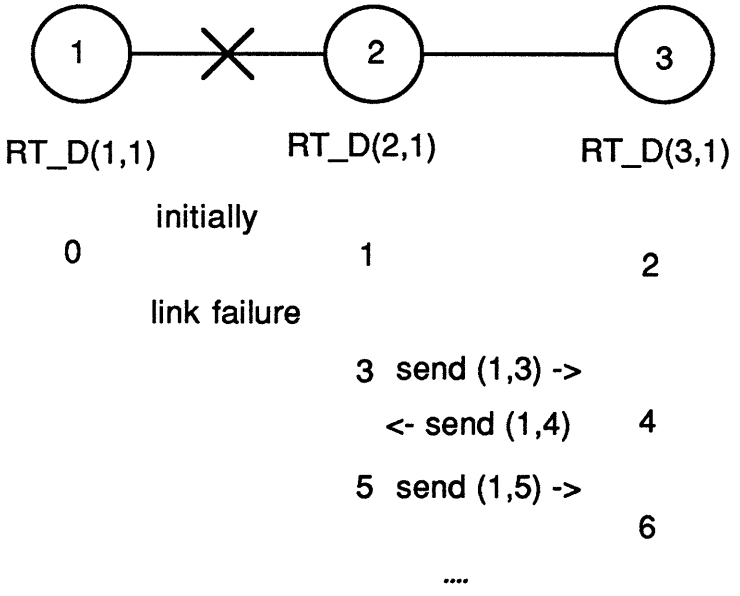


Figure 1: Looping in Ford-Bellman

Figure 2: pseudo code for the algorithm

PART 1

Initialization of node I

```
RT_D(I,I) = 0;
```

Node I detects link P comes Up

```
for each node J NT_D(I,J,P) = 00 ;
send on P a packet containing records (J,RT_D(I,J),RT_N(I,J)) for all nodes J ;
```

Node I detects a change in LEN(P) for link P (LEN(P) = 00 if P is Down)

```
within a finite time COMPUTE();
```

Node I receives an update packet from neighbor M on link P

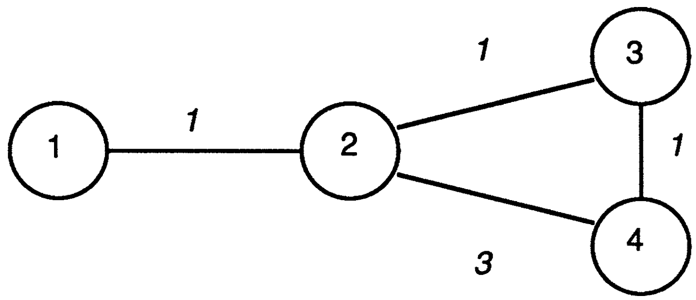
The packet is composed of records (J,D,K), where J is a node, D is a distance and K is a node.

```
for each record (J,D,K) in the packet {
        NT_D(I,J,P)=D;
        if ( J == M) NT_N(I,J,P)= I;
        else         NT_N(I,J,P)= K;
        }
within a finite time COMPUTE();
```

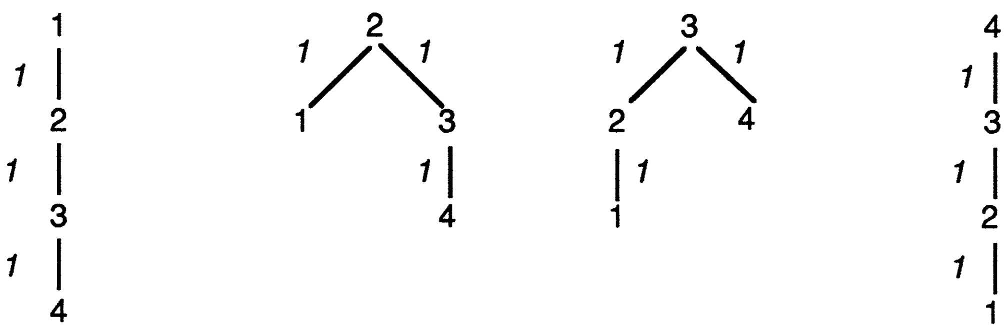
PART 2

COMPUTE() at node I

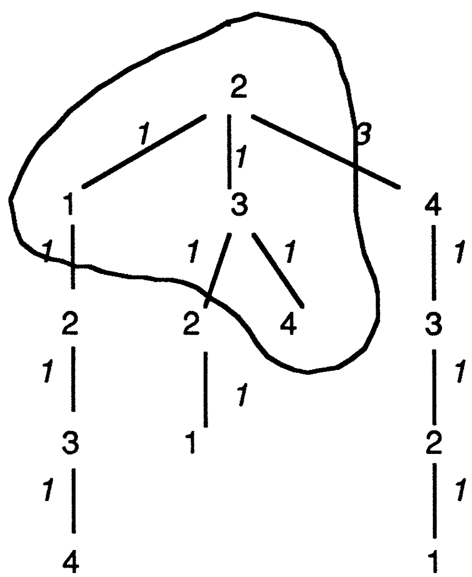
```
for all nodes J UNSEEN(J)= TRUE;
UNSEEN(I) = FALSE;
PACKET = nil;
For each adjacent link P, list the nodes J in order of nondecreasing NT_D(I,J,P). Let TOP(P) denote the
element currently at the top of the list for link P.
While any list is not empty {
        P* = argmin(over P) NT_D(I,TOP(P),P) + LEN(P);
        J = TOP(P*);
        Remove J from list P*;
        if   ( UNSEEN(J)  && ((NT_N(I,J,P*) == I) || (RT_L(I,NT_N(I,J,P*)) == P* ))) {
                if ( RT_D(I,J) != NT_D(I,J,P*) + LEN(P*) ||  RT_N(I,J) != NT_N(I,J,P*)) {
                        RT_D(I,J) = NT_D(I,J,P*) + LEN(P*);
                        RT_N(I,J) = NT_N(I,J,P*);
                        PACKET = PACKET U {(J,RT_D(I,J),RT_N(I,J));
                }
                RT_L(I,J) = P*;
}       }
if ( PACKET != nil ) then send PACKET on all Up adjacent links;
```



a) Network topology

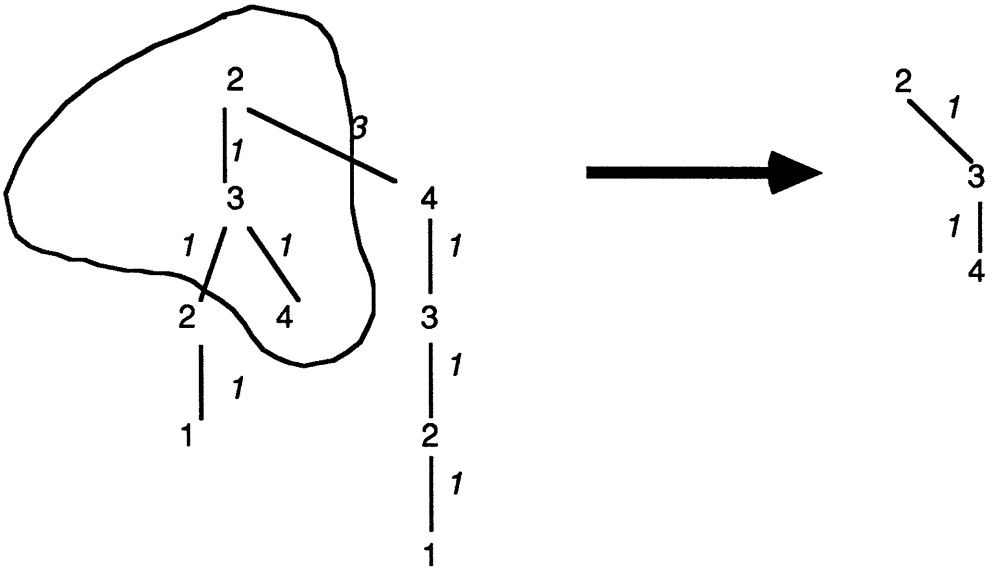


(b) Individual node routing trees



(c) Building the routing tree at node 2

Figure 3: Building the Routing Trees



Building the routing tree at node 2 after failure of link (1,2)
Node 2 realizes at once there is no path to node 1

Figure 4: Reconfiguration following a topology change

REFERENCES

[Bert-87] D. Bertsekas and R.G. Gallager, "Data Networks", Prentice-Hall 1987.

[Burr-81] Burroughs Network Architecture (BNA), Architectural Description, Reference Manual, vol. 1, Burroughs Corp., Detroit, Mich., April 1981.

[Cegr-75] T. Cegrell, "A Routing Procedure for the TIDAS Message-switching Network," IEEE Trans. on Comm., vol. COM-23, no. 6, June 1975, pp. 575-585.

[Dec-83] DECnet, Digital Network Architecture, Routing Layer Functional Specification, Version 2.0.0., Digital Equipment Corp., Maynard, Mass., 1983.

[Dijk-59] E.W. Dijkstra, "A Note on Two Problems in Connection with Graphs," Numerische Mathematik, 1 (1959) pp. 269-271.

[Frie-79] Daniel U. Friedman, "Communication Complexity of Distributed Shortest Path Algorithm", Report LIDS-Th-886, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge MA.

[Garc-87] J.J. Garcia-Luna-Aceves, "A New Minimum-Hop Routing Algorithm", IEEE Infocom '87 Proceedings, 1987, pp. 170-180.

[Gruch-?] S. Gruchevsky and D. Piscitello, "The Burroughs Integrated Adaptive Routing System", Unisys Corp., Southeastern, PA 19398.

[Hago-83] J. Hagouel, "Issues in Routing for Large and Dynamic Networks", Ph.D. Thesis, Graduate School of Art and Sciences, Columbia University, 1983. Also available as IBM Research Report RC 9942.

[Jaff-82] J.M. Jaffe and F.M. Moss, "A Responsive Routing Algorithm for Computer Networks", IEEE Trans. on Communications, Vol. COM-30, No. 7, July 1982, pp. 1758-1762.

[John-84] M.J. Johnson, "Updating Routing Tables after Resource Failure in a Distributed Computer Network", Networks, Vol. 14, No. 3, 1984, pp. 379-392.

[Jubi-85] J. Jubin, "Current Packet Radio Network Protocols", IEEE Infocom '85 Proceedings, Washington, DC, March 1985, pp. 86-92.

[Merl-79] P.M. Merlin and A. Segall, "A Failsafe Distributed Routing Protocol", IEEE Transactions on Comm., Vol. COM-27, No. 9, September 1979, pp. 1280-1288.

[McQu-74] J. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks", Bolt, Beranek and Newman, Inc., BBN Rep. 2831, May 1974.

[Moh-86] H. Mohanty and G.P. Bhattacharjee, "A Distributed Algorithm for the Shortest Path Problem", IEEE Globecom'86, Dec. 1-4, 1986, Houston,Tx.

[Rose-80] E.C. Rosen, "The Updating Protocol of ARPAnet's New Routing Algorithm," Computer Networks, vol. 4, Feb. 1980, pp. 11-19.

[Schw-80] M. Schwartz, "Routing and Flow Control in Data Networks," NATA Advandeced Study Inst.: New Concepts in Multi-user Communications, Norwich, U.K.,Aug. 4-16, 1980; Sijthoof and

Nordhoof, Neth.

[Schw-86] M. Schwartz, Telecommunication Networks: Protocols, Modeling and Analysis, Menlo Park, CA: Addison-Wesley Publishing Co., Chapter 6, 1986.

[Spin-87] J. Spinelli and R.G. Gallager, "Broadcasting Topology Information in Computer Networks", Laboratory for Information and Decision Systems, Report LIDS-1543, M.I.T., Cambridge, MA.

[Spro-81] D.E. Sproule and F. Mellor, "Routing, Flow, and Congestion Control in the Datapac Network," IEEE Trans. on Comm., vol. COM-29, no.4, April 1981, pp. 386-391.

[Taji-77] W.D. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network", Communications of the ACM, Vol. 20, 1977, pp. 477-485.

[West-82] J. Westcott and J. Jubin, "A Distributed Routing Design for a Broadcast Environment", Conference Record of IEEE Military Communications Conference, Boston, MA, October 1982, Vol. 3, pp. 10.4.4-10.4.5.